Multi-agent plan-execution health repair

Jonge, Femke de; Roos, Nico; Herik, Jaap van den

Published in:
EPRINTS-BOOK-TITLE

Multi-agent plan-execution health repair*

Femke de Jonge Nico Roos Jaap van den Herik

IKAT, Universiteit Maastricht,
P.O. Box 616, NL-6200 MD, Maastricht

Abstract

This paper presents a protocol for *plan health repair* in multi-agent plan execution. Plan health repair aims at avoiding conflicts that might arise due to disruptions in the execution of a plan. This can be achieved by adjusting the executions of tasks instead of replanning the tasks. For this purpose, established methods from the domains of planning, discrete event systems, model-based diagnosis, and constraint satisfaction problems have been combined.

1 Introduction

During the plan execution in a multi-agent system conflicts can arise because of discrepancies between the execution and the expectations beforehand. We distinguish two types of conflicts, viz. conflicts that can be repaired by changing the plans (replanning) and conflicts that can be solved by adjusting the execution of the plans within the margins set by the plans. In this article we focus on the second type of conflicts that can arise when execution of a plan cannot be planned in full detail and the smallest parts of a plan, which we will call tasks, are reactive processes by nature. The following example illustrates this type of conflicts among reactive tasks.

Consider a small airport with only one runway used for both arrival and departure. It is a small but busy airport, so plans are tight. Assume that two aircraft agents, agent A and agent B, have agreed on the plans to let B land before A takes off. Here, the smallest planned actions are the departure and arrival of the aircraft, which we will refer to as tasks. The execution of such tasks is a reactive process in the sense that (sub)actions during the execution of a task (for example manoeuvring or changing speed) are not part of the planning. Although the plan is agreed on, still, small changes in the execution (mostly caused by external influences) can cause conflicts. For instance, assume that A is a bit early as the aircraft speeded up while taxiing, and B is a bit delayed because of heavy head wind during its flight. This might cause a conflict because of violating the security constraints on the distance that should be kept between two aircraft. We would like to detect such conflicts as early as possible, so that the agents can agree on a set of reactive actions (repairs) that will prevent the conflicts to occur in the future. For instance, agent A could wait a while before take off. Note that these reactive adjustments are not typified as replanning since the changes in executions will remain within the margins of the planned tasks. Instead, the health of a plan is restored by adjustments in the execution of tasks. We will call this *plan-execution health repair*.

*This research is supported by the Technology Foundation STW, applied science division of NWO and the technology programme of the Ministry of Economic Affairs.

Below, we will introduce a model for a multi-agent system for plan health repair. Based on this model, agents can detect (future) conflicts in advance. Next to presenting such a model, our goals are to apply diagnosis on the model and the anticipated conflicts, and to present a protocol for regaining plan-execution health with a minimal number of repairs.

We model the reactive nature of tasks by introducing set of states for each task and a set of corresponding events that cause state changes within a task. Diagnosing conflicts in this model is related to diagnosis on Discrete Event Systems (DES), e.g. [3], in which agents try to determine occurrence of fault-states based on a partially observable sequence of events. Diagnosis on DES differs from our approach with respect to the diagnostic task. We focus on determining combinations of events that cause violations of constraints between states of different tasks and determining events causing state changes that resolve the constrain violations. In this respect our approach is closer related to Model-Based Diagnosis (MBD). For an overview of MBD in a 'single agent context', see [7]. More recently, MBD has also been studied in a multi- agent context [10]. To our best knowledge, MBD has not been applied to plan health repair, only to diagnosis of planners [1]. Other diagnostic approaches in a multi-agent context we would like to mention are Social Diagnosis, for finding causes for social disruptions in multi-agent systems [6], and diagnosis based on a causal model for achieving multi-agent adaptability [5].

Our plan health repair approach is related to plan repair through replanning [11, 12] and to the TÆMS task descriptions used by Raja et al. [8] for handling uncertainty in plan execution. Moreover, plan health repair can be viewed as a part of distributed continual planning, which addresses the adaptation of plans during plan execution in a multi agent system (for an overview, see [4]). What sets our approach apart is the application of health repair within the margins of the current plans.

The outline of this paper is as follows. In section 2, we introduce a model that agents can use to represent a multi-agent system for plan-execution health repair. Based on this model, agents are able to detect whether conflicts occur in the future. In section 3, we define diagnoses that agents can apply on the conflicts in order to find out what has caused them. A formal definition of both weak and strong plan-execution health repair is given in section 4. In section 5 we present a protocol for agents to regain plan-execution health based on a transformation to a constraint satisfaction problem. In section 6 we provide conclusions and suggest future work on diagnosing and resolving conflict in multi-agent system for plan-execution health repair.

2 Model description

A multi-agent plan $MAP = (A, PD, Cst)$ consists of a set of agents A, a set of plan descriptions PD, containing one plan description for each agent: $PD = \bigcup_{i=1}^{|A|} PD_i$, and a set of constraints Cst between the agents' plans.

A *plan description* $PD_i = (P_i, \mathcal{S}_i, \mathcal{E}_i, \tau_i, \sigma_i, R)$ describes how the plan of agent i will be executed. The base of the plan description is the sequence of tasks $P_i = \langle t_{i,0}, t_{i,1}, ..., t_{i,n} \rangle$ which the agent wants to execute in this specific order. We use $\overline{P_i}$ to denote the corresponding set of all tasks in sequence P_i. To describe the health of each task, the sets $\mathcal{S}_i$ and $\mathcal{E}_i$ contain for each task a set of states and a set of events respectively.

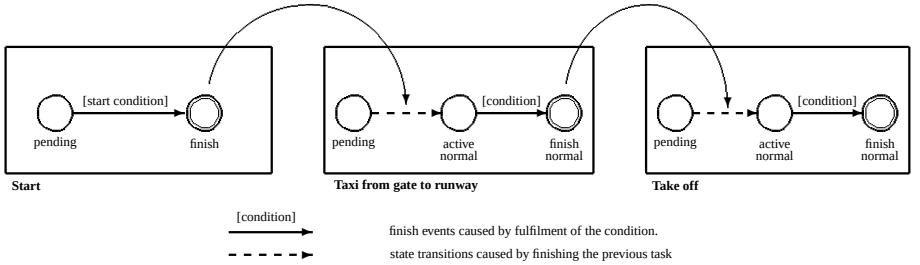

Figure 1: Normal plan execution of agent A.

The functions τ_i and σ_i, and the common rules R formalize the execution of tasks within a plan (we will specify this further on).

During the execution of a plan, a task is in a certain *state*. Each task has its own set of possible states: $S_{i,j} \in \mathcal{S}_i$. The state of a task is denoted by the predicate $ts(t_{i,j}, s)$, with $t_{i,j} \in \overline{P_i}$ and $s \in S_{i,j}$. We distinguish three types of states: pending, active, and finish states. For each task $t_{i,j}$ holds: $S_{i,j} = \{s_{i,j}^{pending}\} \cup S_{i,j}^{active} \cup S_{i,j}^{finish}$. There is only one pending state for each task, this is the state in which the task is awaiting before it is being executed. By finishing the previous task, the next task will become active by changing from the pending to an active state (which state that is, depends on the execution of the previous task). Finally, when the task is completed, the task changes from an active state to a finish state and consequently, the next task is triggered.

Each plan has one start task: $t_{i,0}$, with $S_{i,0} = \{s_{i,0}^{pending}, s_{i,0}^{finish}\}$. The start task has only one pending and one finish state. When the start conditions are fulfilled, this start task will change from the pending to the finish state, which will cause the next task to begin execution (viz. go from the pending state to an active state).

State changes are caused by *events*. Each task $t_{i,j}$ has its own set of events: $E_{i,j} \in \mathcal{E}_i$, with $E_{i,j} = E_{i,j}^{finish} \cup E_{i,j}^{disrupt} \cup E_{i,j}^{repair}$. Finish events are triggered when pre-defined conditions are fulfilled and change tasks from an active to a finish state. Disruption events are externally caused and represent unexpected changes in the execution of a task that might effect the plan-execution health. And finally, the repair events are executed by the agent to regain the plan-execution health when necessary. A tasks state $ts(t_{i,j}, s)$ is the result of the sequence of events $E = \langle e_1, ..., e_k \rangle$ during the plan execution, and will be represented by $[E]ts(t_{i,j}, s)$, where $[E]$ is a modal operator denoting the events leading to $ts(t_{i,j}, s)$. We use $\Box ts(t, s)$ to denote that $ts(t, s)$ will be achieved during the actual plan execution: i.e. the past, current and expected events lead to $ts(t, s)$.

Figure 1 illustrates the normal execution of a plan of the departing agent A in the example of section 1. The plan consists of three tasks: $P_1 = \langle t_{1,\text{Start}}, t_{1,\text{Taxi}}, t_{1,\text{Take\_off}} \rangle$, and has event sequence $\langle e_{\text{finish\_Start}}, e_{\text{finish\_Taxi}}, e_{\text{finish\_Take\_off}} \rangle$.

We model the *execution of a tasks* within a plan by partial functions τ_i and σ_i, and the set of common rules R. The partial function τ_i maps a state and an event to a new state: $\tau_i : \overline{P_i} \times \mathcal{S}_i \times \mathcal{E}_i \nrightarrow \mathcal{S}_i$. τ_i is defined such that only events in $E_{i,j}$ can change the state of a task $t_{i,j}$ into a new state in $S_{i,j}$. We assume that there is exactly one finish event for each task. A task can, by definition of τ_i, reach different finish states depending on de previous state the task is in. The partial function σ_i returns the new state in the next task based on the finish state of the previous task: $\sigma_i : \overline{P_i} \times \mathcal{S}_i^{finish} \nrightarrow \mathcal{S}_i$.

There are three common rules in R which are the same for all agents. The first rule in

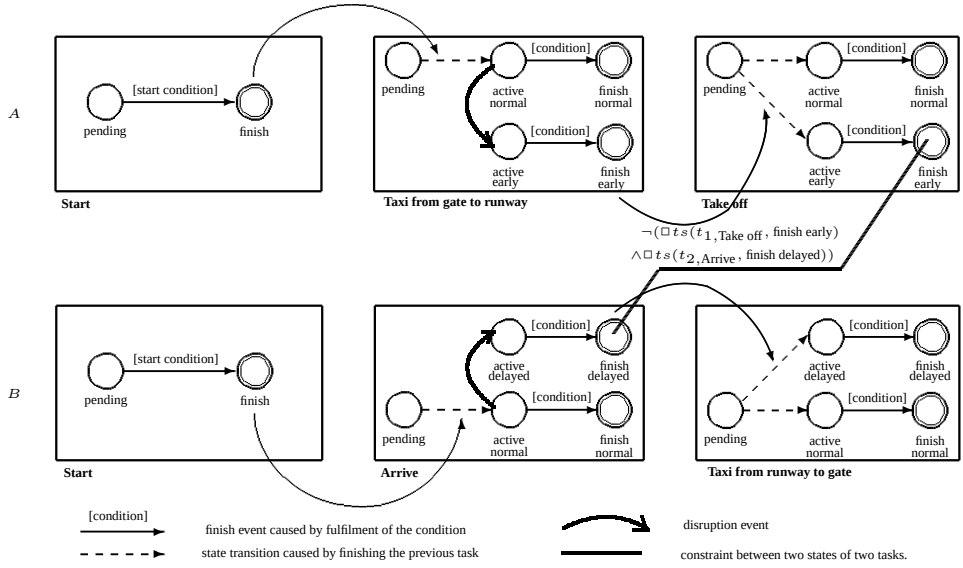

Figure 2: Disturbed plan executions of agents A and B.

R describes how a state transition of a task is caused by an event e_k:

$$([e_1;...;e_{k-1}]ts(t_{i,j},s) \wedge \tau_i(t_{i,j},s,e_k) = s') \rightarrow [e_1;...;e_k]ts(t_{i,j},s') \quad (1)$$

The second rule in R describes the activation of the next task when the previous task is finished:

$$([e_1;...;e_k]ts(t_{i,j},\text{pending}) \wedge \sigma_i(t_{i,j-1},s) = s') \rightarrow [e_1;...;e_k]ts(t_{i,j},s') \quad (2)$$

The third rule in R defines which states will or will not be reached during the plan execution. We use the predicate $Events(\{E_1,....,E_m\})$ to denote that these sequences of events will occur (a sequence E_i for each P_i).

$$\exists e_1;...;e_k(Events(\{\langle e_1;...;e_k;...e_n\rangle_i,...\}) \wedge [e_1;...;e_k]ts(t_{i,j},s)) \leftrightarrow \Box ts(t_{i,j},s) \quad (3)$$

The set Cst in MAP is the set of *constraints*, with each constraint composed of predicates $\Box ts(,)$ and logic symbols $\{\vee,\wedge,\neg\}$. Moreover, constraints are only defined on finish states, as they can be viewed as a summary of the execution of a task. the constraint violations can be repaired during the active states of a task. An example of a constraint is $cst = \neg(\Box ts(t,s) \wedge \Box ts(t',s')) \vee \Box ts(t'',s'')$, in which s, s', s'' are finish states. The constraints are 'demands' on the plan execution that should be fulfilled. A constraint violation or conflict occurs when the expected execution is inconsistent with a certain constraint. We will assume that when plans are executed normally (only finish events occur), all constraints will hold and the plan-execution is in good health. Consequently, the constraint violations are caused by disruption events, and might be solved by repair events to regain the plan-execution health. In addition, we assume that the constraints represent all interdependencies that exist between plans of different agents.

Figure 2 illustrates a disrupted execution of the plans of the departing agent A and arriving agent B in the example of section 1. Both plans consist of three tasks: $P_1 =$

$\langle t_{1,\text{Start}}, t_{1,\text{Taxi}}, t_{1,\text{Take\_off}}\rangle$, and $P_2 = \langle t_{2,\text{Start}}, t_{2,\text{Arrive}}, t_{2,\text{Taxi}}\rangle$. The event sequences of the plan execution are $\langle e_{\text{finish\_Start}}, e_{\text{Speeded\_up}}, e_{\text{finish\_Taxi}}, e_{\text{finish\_Take\_off}}\rangle_1$ and $\langle e_{\text{finish\_Start}}, e_{\text{Heavy\_head\_wind}}, e_{\text{finish\_Arrive}}, e_{\text{finish\_Taxi}}\rangle_2$. In this setting, the constraint $\neg(\Box ts(t_{1,\text{Take\_off}}, \text{finish\_early}) \wedge \Box ts(t_{2,\text{Arrive}}, \text{finish\_delayed}))$ between the two plans is violated.

In general, we assume that each agent has knowledge of its individual plan description PD_i, of the constraints $Cst_i \subseteq Cst$ that are relevant for its plan, and of the other agents to whose plans the constraints Cst_i apply. During the execution of a plan, an agent notices when disruption events occur (for instance through its sensors). Based on this, an agent can construct the sequence of past events (up to and including the current or latest events) in the so-called current event history CEH_i (with $CEH = \bigcup_i CEH_i$). We assume that in the future, from current task $t_{i,j}$ on, no disruption or repair events will occur. Hence, for each task in the remaining plan, one finish event will occur. The resulting sequence of events $FE_i = \langle e_j, e_{j+1}, ..., e_n\rangle$, with $e \in E_{finish}$ and $FE = \bigcup_i FE_i$, will be called the future event sequence. The current event history can be combined with the future events sequence into the future event history: $FEH_i = CEH_i \circ FE_i$ (with $\circ$ denoting a concatenation of the two sequences). Using the future event history, an agent can determine the possible consequences of the disruption events. If the constraints possibly get violated, the agent contacts the other agents involved to verify this. This way, conflicts are detected as early as possible.

3 Event diagnosis

Through diagnosis we wish to find out for each violated constraint, which set of disruption events causes the violation. With the help of these events we can also establish which states are responsible for the violations. These states might be helpful to prevent new constraint violations in much earlier phases by recognizing state patterns.

We define two types of Event Diagnosis: Responsible Event Diagnosis (Δ_R) and Constraint Satisfaction Event Diagnosis (Δ_{CS}). Both diagnoses are a subset of the disruption events that occur in the future event history; $\Delta \subseteq E_{disrupt} \cap \overline{FEH}$, with $\overline{FEH}$ the corresponding set of all events in the sequences in FEH. Moreover, both diagnoses are defined on one constraint cst^* that is violated, but can as well be extended to sets of violated constraints. Below, we provide two formal definitions of the diagnoses. Responsible event diagnosis gives us a minimal set of disruption events that causes the violated constraint cst^*. To achieve this, we remove as many disruption events as possible, such that the violation of the constraint still holds.

Definition 1 *Responsible Event Diagnosis is a minimal diagnosis* $\Delta_R \subseteq \{E_{disrupt} \cap \overline{FEH}\}$ *s.t.* $Events(FEH - \{E_{disrupt} \setminus \Delta_R\}) \cup PD \vdash \neg cst^*$.

With $FEH - X$ we denote that the events in X are removed from the sequences in FEH.

The constraint satisfaction event diagnosis gives us a minimal set of disruption events, such that when these events are left out of the future events history, the violated constraint will hold again.

Definition 2 *Constraint Satisfaction Event Diagnosis is a minimal diagnosis* $\Delta_{CS} \subseteq \{E_{disruption} \cap \overline{FEH}\}$ *s.t.* $Events(FEH - \Delta_{CS}) \cup PD \vdash cst^*$.

The two diagnoses are related as follows: each minimal hitting set on the set of all possible responsible event diagnoses, is a constraint satisfaction diagnosis, and vice versa.

In the example of section 1, all possible event diagnoses are:
$\Delta_R = \{e_{\text{Speeded\_up}}, e_{\text{Heavy\_head\_wind}}\}$, $\Delta^1_{CS} = \{e_{\text{Speeded\_up}}\}$ and $\Delta^2_{CS} = \{e_{\text{Heavy\_head\_wind}}\}$.

4 Plan-execution health repair

Once the agents have detected the constraint violations that will arise because of (some of) the occurred disruption events, the agents should adjust the execution of the plans such that no constraint violations will occur in the future and the plan-execution health is restored. To achieve this, each agent can insert repair events in the future event history in order to create new state paths in its plan execution. By inserting repair events, the anticipated constraint violations can be avoided.

A weak plan-execution health repair FER^- is a set of event sequences containing all future event sequences with some repair events inserted, such that by applying FER^-, all anticipated constraint violations will dissolve and no new violations will be created.

Definition 3 *A weak plan-execution health repair FER^- is a set $FER^- = FE \Cup RE$, where RE is a minimal subset of E_{repair} s.t. $Events(CEH \circ FER^-) \cup PD \cup Cst \not\vdash \bot$.*

We use $FER^- = FE \Cup RE$ to denote that the events in RE are placed at specified places within the sequences collected in FE. Note that for the same FE and RE different sets $FER^- = FE \Cup RE$ are possible, depending on the placement of the repair events in the sequences in FE. With a minimal RE we limit the subsets of RE to those which have no subset that will construct a (weak) plan-execution health repair as well.

A strong plan-execution health repair FER^+ differs from the weak version in that FER^+ ensures that all constraints hold.

Definition 4 *A strong plan-execution health repair FER^+ is a set $FER^+ = FE \Cup RE$ where RE is a minimal subset of E_{repair} s.t. $Events(CEH \circ FER^+) \cup PD \vdash Cst$.*

Proposition 1 *A weak plan-execution health repair is a strong one and vice versa.*[†]

In the example of section 1, we can introduce an event e_{Wait}, which changes $ts(t_{1,\text{Take\_off}}, \text{active\_early})$ into $ts(t_{1,\text{Take\_off}}, \text{active\_normal})$. Then, an example of a plan execution health repair is $FER = \{\langle e_{\text{fin\_Start}}, e_{\text{Speeded\_up}}, e_{\text{fin\_Taxi}}, e_{\text{Wait}}, e_{\text{fin\_Take\_off}}\rangle_1, \langle e_{\text{fin\_Start}}, e_{\text{Heavy\_head\_wind}}, e_{\text{fin\_Arrive}}, e_{\text{fin\_Taxi}}\rangle_2\}$.

5 A protocol for plan-execution health repair

Both the weak and strong plan-execution health repair are strongly related to model-based diagnosis. The weak plan-execution health repair corresponds with consistency-based diagnosis, as formalized by Reiter [9]. The strong plan-execution health repair is a type of abductive diagnosis, as defined by Console and Torasso [2]. Since both types of diagnosis are known to be NP-hard, in general, our plan-execution health repair is NP-hard as well.

[†]The proof is omitted because of limited space. It is available on request.

To enable the agents to find a plan-execution health repair, we formulate the plan-execution health repair as a constraint satisfaction problem: $PR_{csp} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$. The set variables $\mathcal{V}$ contains a variable for each task in MAP: $\mathcal{V} = \{v_{i,j} | t_{i,j} \in \overline{P_i}\}$. $\mathcal{D}$ contains for each variable a domain of possible values, in this case the set of finish states: $\mathcal{D} = \{S_{i,j}^{finish} | t_{i,j} \in \overline{P_i}\}$. The set of constraints, $\mathcal{C}$, is divided into plan constraints, $\mathcal{C}_{plan}$, and conflict constraints, $\mathcal{C}_{conflict}$. The plan constraints represent the execution of the plans, as described by PD. A plan constraint between two successive tasks is true, if there is an event path from the value assignment (or finish state) of the first task, to the value assignment (or finish state) of the second task. The possible paths depend on CEH and FER (the future event sequences combined with repair events). Therefore, the set of plan constraints can be constructed as follows.

$$\mathcal{C}_{plan} = \{c_{v_{i,j}, v_{i,j+1}}(s_1, s_2) | Events(CEH \circ FER) \cup PD \vdash \Box ts(t_{i,j}, s_1) \wedge \Box ts(t_{i,j+1}, s_2)\} \quad (4)$$

The set of conflict constraints $\mathcal{C}_{conflict}$ is a direct mapping of the set Cst in MAP onto the variables $v_{i,j}$.

$$\mathcal{C}_{conflict} = \{c_{v_{i,j},....,v_{k,l}}(s_1, ..., s_p) | \Box ts(t_{i,j}, s_1) \wedge .... \wedge \Box ts(t_{k,l}, s_p) \vdash cst\} \quad (5)$$

We can build a constraint graph by representing the variables by nodes and the constraints by hyper-arcs. Based on this graph, we present the following 3-stage multi-agent protocol for finding a plan-execution health repair after detecting violated constraints.

Stage 1: Initially, the agents attempt to solve the violated constraints locally. Therefore, all agents that are not involved in a constraint violation, lock the values of their variables. Moreover, all values of variables that have their tasks in the past are locked. Then, each agent creates a linear individual constraint graph for its own plan. Influences of other agents' plans through the conflict constraints will be represented by unary constraints on the nodes involved. The changes in domains based on the locked variables are updated in the graph as well.

Stage 2: by repeating two steps, arc-consistency on the constraint graph is achieved. First, the agents reduce the domains of their variables by applying arc-consistency on their individual graphs. To achieve this, no communication with other agents is required. Since the individual graphs are linear, arc-consistency will be achieved in linear time.

Second, for each constraint in MAP in which the agent is involved and for which the related variables have an altered domain, each agent communicates the new domain to the other agents involved in the constraint. Subsequently, the agents adjust the unary constraints representing the influences of the plan-constraints. The two steps are repeated until no domains change anymore.

Stage 3: based on the restricted domains, the agents can search for a value assignment. First, they agree on a order in which the agents will search for an assignment. Then, the first agent in the order searches for a value assignment for its variables and communicates these to the agents involved in its conflict constraints. These agents adjust their graphs based on this knowledge and thereupon, all agents apply arc-consistency on their adjusted graphs, as described above. If this succeeds, the second agent in the order searches for a value assignment, and so on. But when during the arc-consistency procedure a domain becomes empty, the search process backtracks and the previous agent has to find a new assignment. Eventually, a solution for the constraint satisfaction problem may be found and subsequently a plan-execution health repair can be established.

When no solution is found, the search space can be increased through unlocking some of the locked variables. Another alternative or last resort is to apply replanning.

It is possible to change the third stage of the algorithm such that the assignment with the minimal number of repair events is found. In addition to this, agents can be enabled to negotiate on which agent should perform which repairs.

6 Conclusions and future work

In this paper, we introduced a model for reactive execution of plans in a multi-agent context. The model forms a basis for adequate handling of conflicts that can arise during the execution of the plans. Using this model, agents may perform appropriately model-based diagnosis to resolve conflicts and regain plan-execution health by inserting small repair actions (the repair events) in the execution of the tasks.

Still, several topics are to be examined in the future. First, the model should be extended to a probabilistic model in which the chances that a disruption event will occur in the future are taken into account. Second, the heuristics to improve the efficiency of the described protocol for multi-agent plan-execution health repair should be tested and further investigated.

References

[1] L. Birnbaum, G. Collins, M. Freed, and B. Krulwich. Model-based diagnosis of planning failures. In *AAAI*, pages 318–323, Boston, July 1990.

[2] L. Console and P. Torasso. A spectrum of logical definitions of model-based diagnosis. *Computational Intelligence*, pages 133–141, 1991.

[3] R. Debouk, S. Lafortune, and D. Teneketzis. Coordinated decentralized protocols for failure diagnosis of discrete-event systems. *Journal of Discrete Event Dynamical Systems: Theroy and Application*, 10:33–86, 2000.

[4] M.E. desJardins, E.H. Durfee, Jr. C.L. Ortiz, and M.J. Wolverton. A survey of research in distributed, continual planning. *AI Magazine*, 4:13–22, 2000.

[5] Bryan Horling, Victor Lesser, Regis Vincent, Ana Bazzan, and Ping Xuan. Diagnosis as an Integral Part of Multi-Agent Adaptability. *Proceedings of DARPA Information Survivability Conference and Exposition (see also UMASS CSTR 1999-03)*, pages 211–219, 2000.

[6] M. Kalech and G.A. Kaminka. On the design of social diagnosis algorithms for multi-agent teams. In *IJCAI*, 2003.

[7] I. Mozetic. Model-based diagnosis: an overview. In *Advanced Topics in Artificial Intelligence*, pages 419–430. Springer-Verlag (LNAI 617), 1992.

[8] A. Raja, V. Lesser, and T. Wagner. Towards robust agent control in open environments. In *Proceedings of 5th International Conference of Autonomous Agents*, 2000.

[9] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57–95, 1987.

[10] N. Roos, A. ten Teije, A. Bos, and C. Witteveen. An analysis of multi-agent diagnosis. In *AAMAS*, 2002.

[11] S. Russel and P. Norvig. *Artificial Intelligence, A Modern Approach*. Prentice Hall, 1995.

[12] R. van der Krogt, M. de Weerdt, and C. Witteveen. A resource based framework for planning and replanning. *Web Intelligence and Agent Systems*, 1(3-4), 2003.